A Scalable Auto Tuning Framework using Machine Learning Techniques

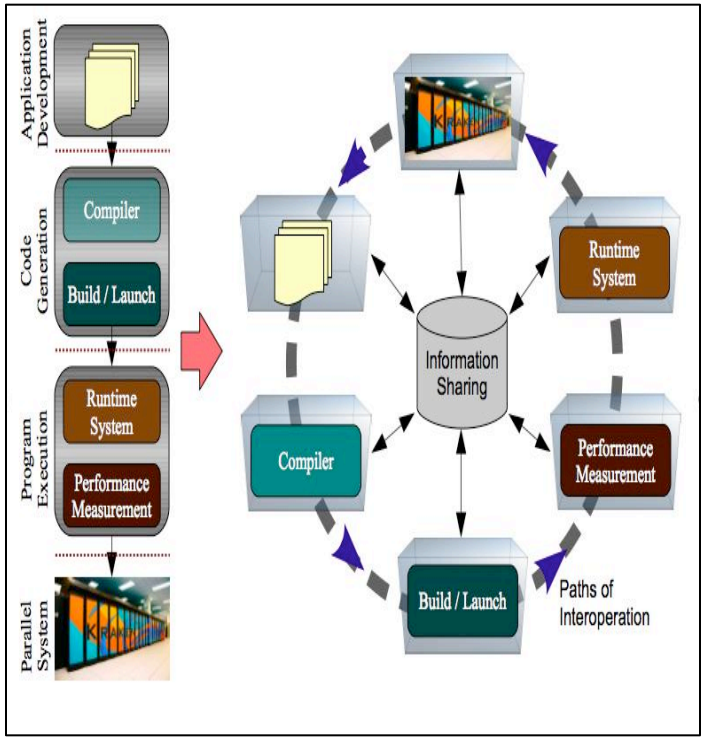
Abid M. Malik, Mohd. Abdullah Bari, and Barbara Chapman
HPCTools Group
University of Houston

Outline

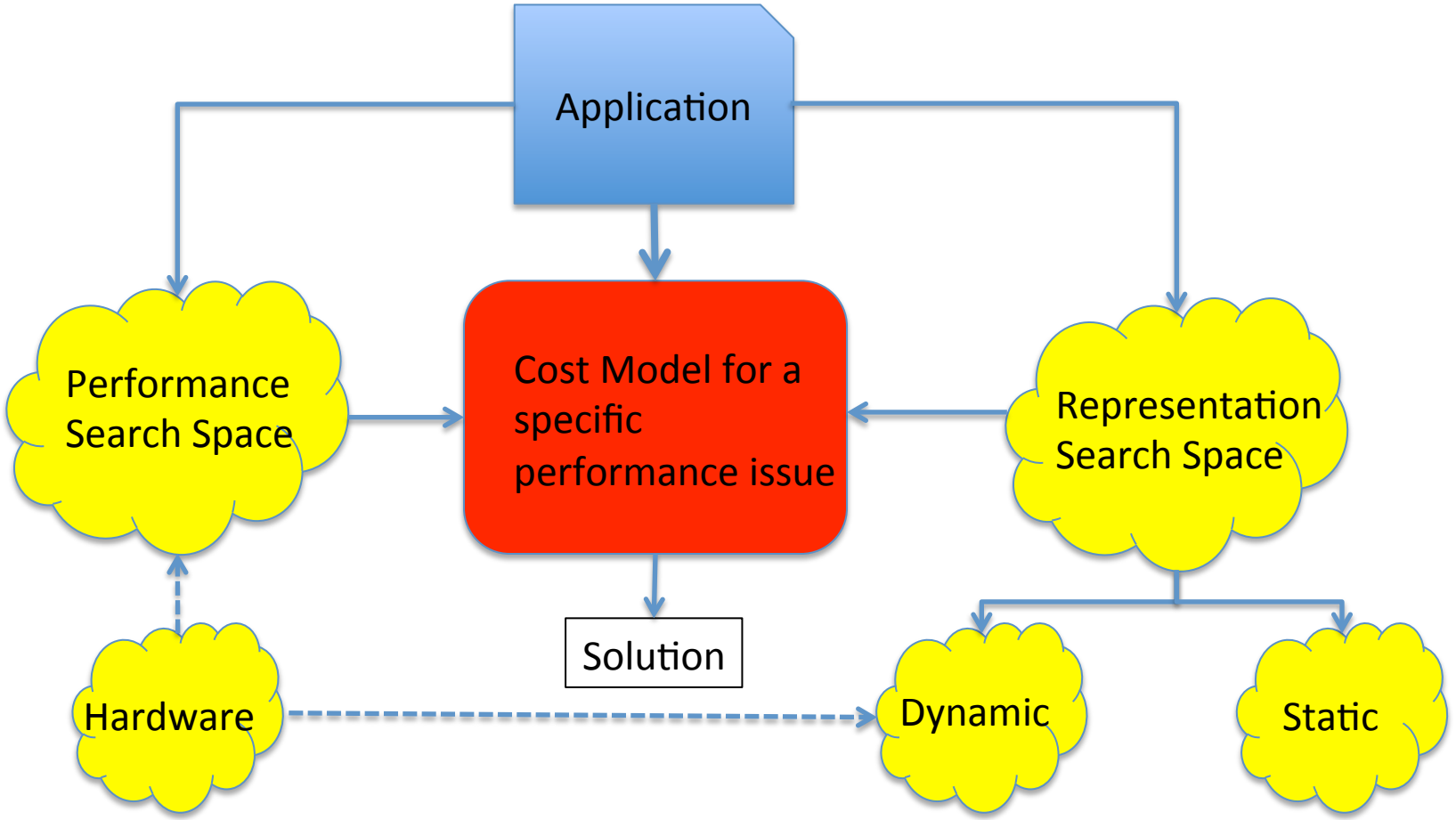
- Introduction
- Our approach
  - Optimization problem
  - Feature selection
  - Machine learning techniques
- Experimentation and results
- Conclusions
- Future work

Introduction

- What is the main challenge?
- Traditional performance analysis approach might not be enough to get optimal performance
- More information is needed
- More tools interaction is needed to get a better and useful information
- Cost modeling for auto tuning will be a big challenge for the future HPC



Introduction



Motivation

- **To get a better cost model**:
  - We need a better search space which ensures that we have an optimal solution (or maximum number of good points)
  - We need a better representation search space which can capture uniqueness of a given kernel, application etc.
  - We need a better technique that can map both spaces effectively to find the **best solution**

Phase ordering problem

- Classical NP-hard problem in compiler
- Number of optimizations in the back end
- *What should be the parametric values for each optimization and in which order they should be applied?*
- Modern compilers use a heuristic which ensures an acceptable performance for most of the applications
- Previous work: mostly playing with the flag switching provided by a compiler. Quality of search space is not complete!
- Data locality optimization: main optimization to gain the performance on modern multi-core processors

Search space for data locality optimization

- The search space for data locality optimization
  - loop unrolling
  - Loop fusion
  - Loop fission
  - Loop interchange
  - Loop tiling/blocking
  - Loop prefetching

Representation search space

- Static features:
  - Number of instructions
  - Number of basic blocks
  - Number of loads instructions
  - Number of store instructions
  - Number of integer instructions
  - Number of floating point instructions
  - Number of loops
  - Level of loops

Representation search space

- Dynamic features
  - L1 cache miss rate
  - L2 cache miss rate
  - L3 cache miss rate
  - TLB miss rate
  - Average cycles per instruction
  - Average branch taken
  - Average conditional branch per branch instruction
  - Average branch instruction per instruction
  - Average stall cycles per instruction

Machine Learning techniques

- Analytical Models
  - description of a system using mathematical concepts and language
  - Low adaptability
- Supervised learning
  - each example is a *pair* consisting of an input object (typically a vector) and a desired output value (also called the *supervisory signal*)
- Support Vector Machine (SVM)
- Artificial Neural Network (ANN)

Machine learning techniques

- Empirical Model: run each point in the performance search space to get the best solution
- Give better solution, adaptable
- Very high computational cost
- Clustering / Unsupervised learning
- KNN (K-Nearest Neighborhood) clustering approach

Feature selection criteria

- Information gain
- Mutual Information
- Chi-statistic
- Frequency of feature

Classification problem

- Parameter value for each optimization is a class
- Loop unrolling : number of time it can be rolled
- In which order the optimization is applied
- We have two classifiers: one for finding the parameter value for an optimization strategy and the other find the order in a set of optimizations

Experimentation and Results

- Hardware: Intel Xeon
- Compiler: OpenUH, Intel, GCC, and PGI
- CHiLL framework to produce the search space
- Wika machine learning tool
- Benchmark: Poly-bench

Experiment (Using SVM)

- Non-linear SVM
- Accuracy more than 90%
- Improvement in performance from 10% to 35%

Benchmark	OpenUH	ICC	GCC	PGI
2mm	10	3	7	2
3mm	11	4	12	3
adi	11	5	9	4
atax	12	5	13	6
bicg	13	6	10	7
cholesky	15	4	16	7
correlation	28	6	18	8
coveriance	10	7	11	8
doitgen	13	9	10	7
durbin	24	3	18	6
dynprog	15	4	10	5
fdtd-2d	14	4	9	8
fdtd-apml	21	5	15	8
gauss-filter	12	7	10	9
gemm	20	3	15	10
gemver	11	8	9	7
gesummv	10	3	7	6
gramschm	10	3	9	7
lu	20	10	13	7
ludcmp	25	8	17	8
mvt	10	3	7	3
reg-detect	25	10	20	10
seidel	30	15	22	11
symm	35	13	20	11
syr2k	19	10	18	9
syrk	20	10	22	9
trisolv	15	8	10	9
trmm	25	10	20	10

Experiment (Using Clustering)

- Able to reduce the search space for Empirical models
- Performance improvement from 7% to 20%

Cluster 1	Cluster 2	Cluster 3
2mm	adi	fdtd-2d
3mm	atax	fdtd-apml
symm	bicg	gauss-filter
syr2k	cholesky	gemm
syrk	correlation	gemver
trisolv	coveriance	gesummv
trmm	doitgen	gramschm
	durbin	lu
	dynprog	ludcmp
	mvt	
	reg-detect	
	seidel	

Hybrid Approach

- Combining supervised and unsupervised learning
- Clustering and then training SVM for each cluster
- Accuracy performance of SVM increased by 2%
- The performance increased from 2% to 4%

Conclusions

- Machine learning techniques have great potential to solve various tuning problems for the future HPC
- Our results strengthen this belief
- Main challenges
  - Efficient performance search space
  - Better feature selection techniques
  - Modeling the performance problem
  - Correct selection of a learning methodology

Future work

- Auto tuning for power performance using machine learning
- Runtime/dynamic tuning strategy for various runtime libraries
- Software adaptation: predicting which version of a code will give better performance under certain environment
- Learning models for other HPC performance issues, e.g., resilience, porting , fault tolerance etc.

Questions!!